

Supporting Document Development with Concordia

Janet H. Walker

Symbolics, Inc.

“Documentation projects are always behind schedule and over budget, and the quality of the final result is disappointing.” “Documentation is a problem that everybody knows about and wishes would just go away.” “Nobody ever really wants to write technical documentation; it’s the boring afterthought after all the hard work is finished.”

What causes this kind of popular perception about technical documentation? Why is documentation the unglamorous part of most projects? Could it be due to a lack of appropriate technology for supporting the job? While software developers have long produced software development environments for themselves, few people have addressed the analogous working environment issues for technical writers.^{1,2}

It is surprising that software development environments and document development environments have remained quite separate, since writing code and documenting it are really the same kind of effort at some abstract level. Document engineering is no less an intellectual challenge than software engineering, and certainly offers a greater operational

Concordia applies object-oriented techniques to creating, publishing, and maintaining complex documentation. Using this highly integrated working environment, writers move beyond conventional limits on their productivity.

development paradigm, using the same substrate as the Genera system, and it integrates seamlessly into Genera as a conceptual extension.

Concordia integrates the facilities needed to create, revise, publish, distribute, and maintain very large document sets with very long life cycles. It provides an environment for professional communicators to create and maintain large information bases of text and graphic information, with the capability for large-scale information development and delivery. Various generations of this application have been used in-house at Symbolics since 1984 for producing system software documentation.

Authoring environment. In their daily work writers create, revise, publish, and maintain a document database. Concordia has specialized support for the different phases of a document life cycle: writing, editing, illustration, design, production, and maintenance.

For small jobs, a single person could take on all roles in the process; personal or desktop publishing systems support this model of document production. (For the advantages and disadvantages of desktop publishing in handling documentation, see the sidebar “What about desktop publishing?”) For large documentation jobs, tasks are typically performed by different people at different times or by the same

Another version of this article appears in *Conference Record HICSS-21*, Hawaii International Conference on System Sciences, January 5-8, 1988, Kailua-Kona, Hawaii.

person at different times. Accordingly, Concordia supports the different tasks with their own software facilities. For example, the environment separates specifying book design from editing content. Likewise, it separates production from modification of the document's overall structure.

Despite separate support for each task, a single application integrates all of the environment's capabilities. As writers' tasks change at different times in the document's life cycle, they use different abilities of the application. Thanks to the integration, parts of a developing document are always available when needed, in the format required. Concordia requires no format conversions to move from one kind of task to another.

Goals and design

The Concordia development project had the following broad goals: enhance productivity of technical writers, speed time-to-market for documents, streamline document maintenance processes, reduce maintenance costs, and increase flexibility in document delivery options (including on-line delivery).

Our somewhat abstract productivity goals became more concrete implementation projects: a database for documents,⁴ embodying hypertext⁵ concepts; a structure editor, combining concepts from what-you-see-is-what-you-get editing (called WYSIWYG) and generic markup languages⁶; an object-oriented graphic editor for manipulating illustrations; configuration tools for managing versions and postproduction distribution; incremental development tools to support a group of writers cooperating on the development of a large document set; and a flexible interface for on-line inspection of documents during development and for final delivery to customers.

Architecture. Concordia is part of the Symbolics documentation system, diagrammed in Figure 1. The central component is the documentation itself, arranged in a database of independently accessible records maintained using Concordia. End users can retrieve information from the database using Document Examiner,⁷ the delivery interface for on-line lookup. We also deliver the database as published manuals, in conventional paper form, and could extend to other electronic media.

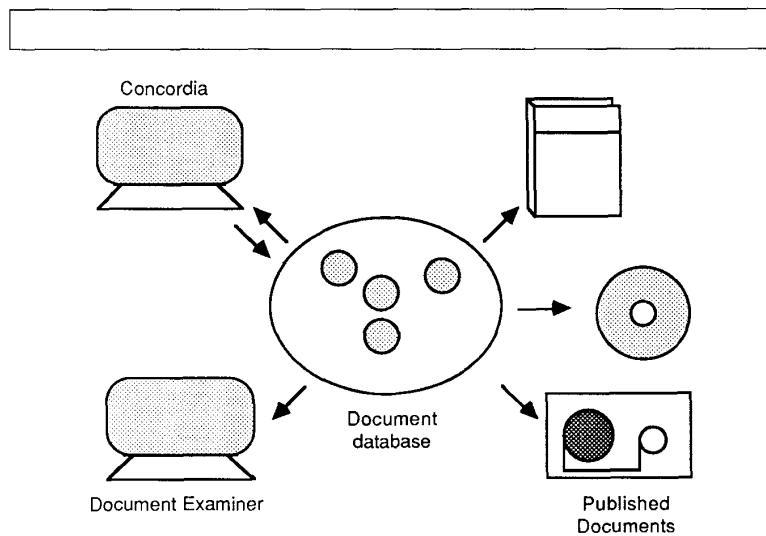


Figure 1. Document system architecture. The central component is the documentation itself, arranged in a database of independent records. This database is written using Concordia. End-user retrieval from the database is handled using Document Examiner, the delivery interface for on-line lookup. The database can be published as conventional paper manuals or in various electronic forms.

What about desktop publishing?

Doesn't desktop publishing provide the key to solving all documentation problems? Desktop publishing has done much to speed up document production, with many of the same goals as Concordia—more productive documentation departments producing more cost-effective publications faster. Effective as it is for the right jobs, desktop publishing is designed for a different kind of document than Concordia.

The differences between desktop publishing and Concordia become apparent only when you examine the documents, their life cycles, and the processes used to produce them. Desktop publishing is designed for short documents (from five to a practical limit of around 50 pages) produced for a single use, often by one person. Concordia was designed for large documents (hundreds or thousands of pages) with very long development and life cycles, maintained by a team of writers.

In a short document, relatively little time goes into preparing the content of the document; most of the effort goes into typesetting, layout, and final production. Desktop publishing moved control over those details into the hands of the people creating the document, thus reducing delays and miscommunication and giving people more control over their own products.

In large documentation projects, the development (writing) stages take over half of the project resources, while final production takes much less (less than 10 percent according to some estimates). Therefore, to improve productivity and reduce costs in large documentation projects, you need to concentrate on the development cycle; relatively small gains come from improving the final production cycle.

Desktop publishing systems differ in the details of their features, but all have essentially the same goal—good layout of text and graphics on paper for subsequent reproduction. Many desktop publishing systems have only a primitive word processor, on the assumption that the actual preparation of the text happens elsewhere. They provide primarily a publishing facility, without features for assisting large-scale development.

Concordia, on the other hand, changes the nature of document development instead of simply speeding up the manual production process.

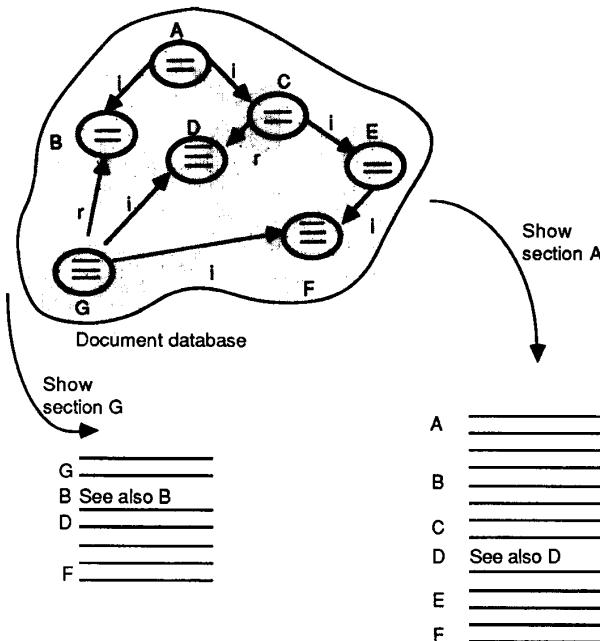


Figure 2. Document structure. Independent records in the database are represented by the ovals labeled A through G. Records can refer to each other by way of links, which have types. The links labeled i are inclusion links, specifying that the target record is included at that point; the links labeled r are conventional cross-reference links. The figure shows what a reader of records A and G would see. The i links have been replaced by the contents of the target records and the r links have been replaced by cross-reference sentences.

Document database. Conceptually, the documentation is organized as a database of interconnected modules called *records*. A record is a unit of information retrievable from the database, as well as a semantic unit of the subject matter. Think of records as semantic cut-and-paste units.

Records contain the subject matter from which we construct documents. A document is formed by linking records together, much as a program is formed by the calling structure of subroutines and functions. Figure 2 diagrams the creation of a document from a set of records by means of links between the records. We use this mechanism for creating conventional hierarchical documents from a collection of independent units. The records

are reusable units and can be used in more than one document.

Writers decompose subject matter into records fairly naturally in most cases, since the divisions depend on the conceptual structure of the subject matter. Reference material and definitions of terminology easily separate into independent records. Conceptual and tutorial materials are harder to organize, much the same as in conventionally written manuals. As a rule of thumb, writers construct a separate record for any item of information that a reader might need to access directly. Thus the process of constructing the records for a document relates to the purpose of the document and the needs of the audience. As with software modularity, it takes expe-

rience and judgment to construct well modularized documents.

This conceptual database is not currently implemented using any standard database technology. Instead, we based our implementation on a set of binary files containing records and indexes. The database consists of any number of document sets (one set for each software product), where each document set contains one or several books. (A good explanation of the concepts of document databases appears in James.⁴) Abstractly, we have books within document sets within the database; physically, we have records within files within a collection of configuration-controlled systems.

End-user delivery interface. We deliver documentation to customers both on paper and on line. On-line delivery is more important because it represents the future direction for information delivery in the computer industry.

On-line delivery of the document database is managed by Document Examiner, a window-based utility closely integrated with the rest of the Symbolics software environment. (See the sidebar "An on-line manual" for further information on Document Examiner.) In the course of developing documentation, writers often use the facilities of Document Examiner to see how a particular section will appear to its readers.

Creating and editing documents with Concordia

Concordia is a framework organizing the tasks and activities in the document life cycle. The three major subactivities are text editing, graphic editing, and previewing.

Its editor is the major software component of Concordia, since the majority of time in a complex project is spent in development, not in proofing, layout, and production. Concordia modifies and extends the standard software editing paradigm of Symbolics Genera to suit large-scale document editing jobs. Figure 3 shows a screen display of Concordia, with the editor visible. The editor is a structure editor, capable of manipulating the records represented within files.⁸

The text editing component of Concordia, called ConEd, provides editing capabilities that are not WYSIWYG yet do not

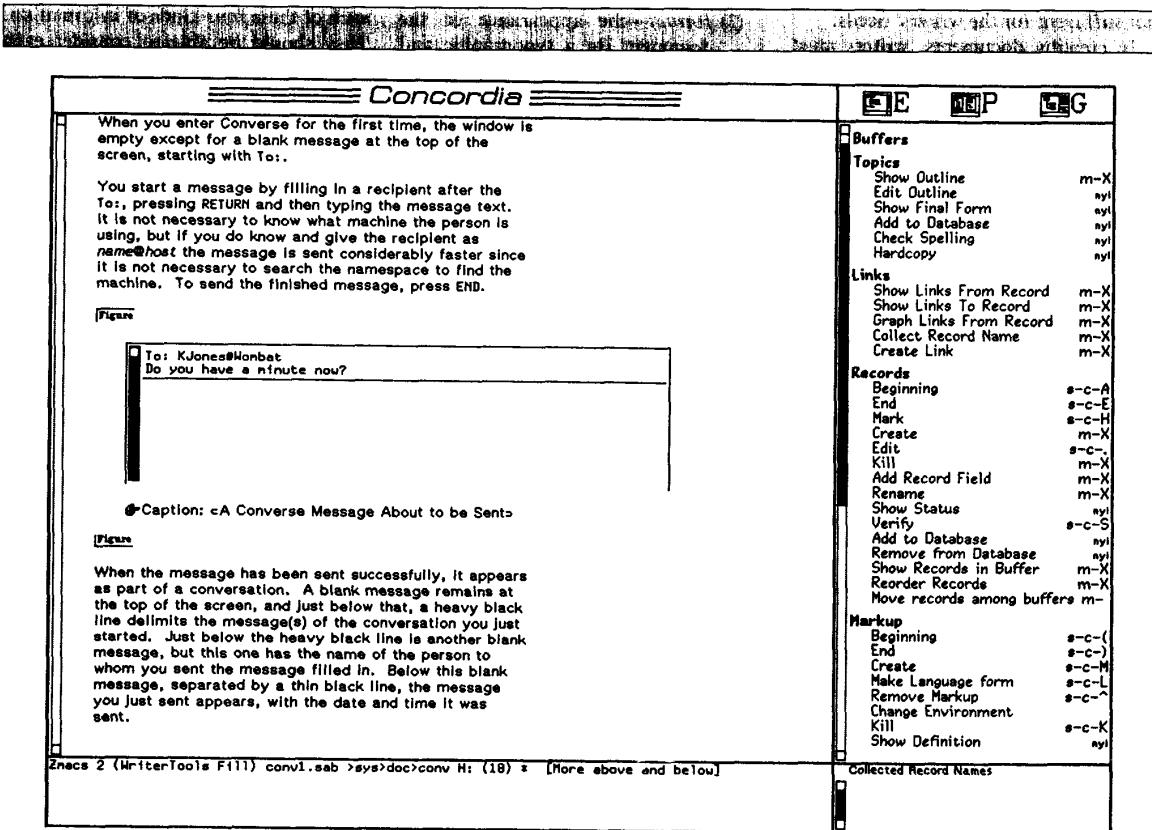


Figure 3. Concordia screen display, showing the ConEd document editor. It shows part of a record in which a figure appears. The right-hand panel is a command menu showing the commands that manipulate record structure and appearance. All of these commands can be issued via keyboard commands as well as by using the menu.

require writers to maintain document sources in a conventional textual markup language. ConEd implements an amalgam of these capabilities that we call *semblance editing*.

WYSIWYG editing requires writers to attend simultaneously to structure, content, and appearance. For small jobs, writers can fail to notice the conceptual differences between these aspects of a document. The emphasis in WYSIWYG editors (in fact, their *raison d'être*) is on controlling the appearance of a document. This partially carries over from the days of typing pools, when the typist's only job was to ensure the good appearance of the text. Writers of large document sets should care little about final appearance during development. What they do care about during development is being able to

categorize the information for display—text, headings, tables, lists of various kinds, and so on.

Concordia uses a generic markup language for documentation because of the importance for writer productivity of separating content from form. The difference between this and other editor/formatter systems based on generic markup is that users do not edit text intermingled with textual commands. (For more on formatting, see the sidebar "Formatter command languages.") Instead they edit a semblance of the final result, as shown in Figure 4.

The figure contains an example of markup, indicated by the small boxed delimiters surrounding a text area on the screen. Itemize is a specification about the communications intent of the enclosed text

(and, eventually, its appearance). The list is indented and set off from the body text much as it would be in a WYSIWYG editor. Unlike WYSIWYG, there is no attempt to show the nature of the highlighting or the exact details of the surrounding spacing. What you can't see from looking at the figure is that the writer typed in only the request for markup, not any of the formatting effects; ConEd made the decisions about spacing and indenting.

Rather than doing real-time formatting, ConEd shows the writer some semblance of the final formatted result. This provides the feel of a WYSIWYG editor without sacrificing any of the power of a generic markup language. ConEd supplies *formatting on demand* of any region of the text, for occasions when the semblance is

not sufficient for the writers' needs.

In creating documents, writers must manage four different kinds of information:

- (1) *structure*—the organization of material, for example, in a chapter/section/subsection hierarchy;
- (2) *content*—the subject matter of the document;

- (3) *format*—the appearance of the document (in a typographic and information design sense); and
- (4) *meta-information*—auxiliary information not generally part of what the reader of the document sees, but relevant nonetheless to its development.

Writers need assistance with managing

each of these four kinds of information. They should be able to consider each aspect of a document relatively independently of the others, and not have to work simultaneously at several levels of discourse.

Structure editing. In Concordia, records are the raw material from which docu-

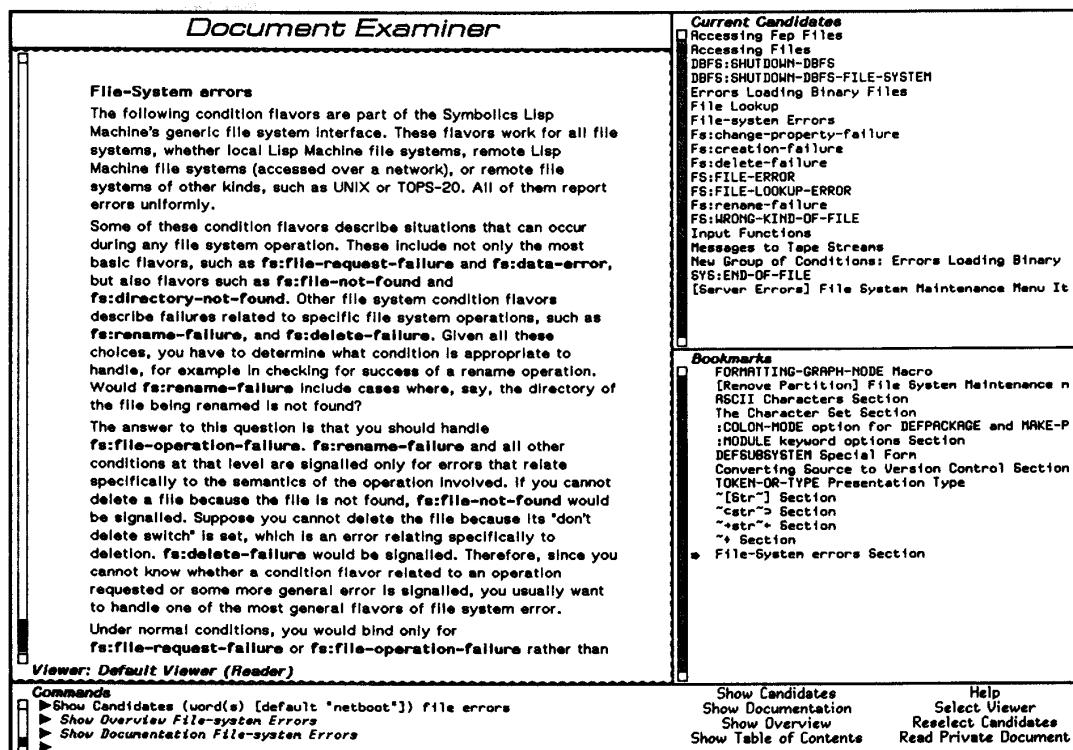
An on-line manual

How important is on-line delivery for documentation? Would anyone read manuals on line if they could have paper instead? With over four years experience behind us, we believe that on-line manuals are highly feasible. We use our manuals heavily on line, with some software engineers reporting that they use the on-line version exclusively. The on-line manual has exactly the same contents as the printed manuals delivered with the system, because both are produced from the same database of records. So people choose the on-line manual based on satisfaction with it, not on differences in content. The key elements of its success are adequate display hardware and a

good user interface.

The interface to our on-line manual, called Document Examiner, is an independent window-based utility closely integrated with the rest of the Symbolics Genera environment. The window is divided into panes (subwindows) that help users manage various aspects of their interaction with the document. (See the accompanying figure on Document Examiner's screen display.)

The majority of the screen area is used to show a topic. It gives people the feeling of reading a section from a book. The bottom area of the screen contains both a fixed command



Document Examiner screen display. The viewer area contains the first screenful of a section, whose bookmark is marked in the bookmarks pane. The candidates pane contains the results of a search for relevant topics. Several recent commands are visible in the command pane.

ments are constructed. Structure is imposed on the raw material by links from one record to another (as sketched in Figure 2). The links are all executable; they tell the formatter or display to include the target record as if it were part of the originating record. Figure 4 shows a record containing literal text as well as a number of links to other records.

The position of a record within a document depends on the links to it. If the top-level record is called a chapter, then the records it has links to would be called sections; the records that those sections link to would be called subsections, and so on. These organizational levels are not part of the records themselves, but rather are assigned dynamically by the process of

viewing the record—from a reader's point of view.

In one sense, a document is a directed graph structure. Editing this structure requires a specialized set of commands for manipulating both records themselves and the links among them.

Creating a record. Since records are

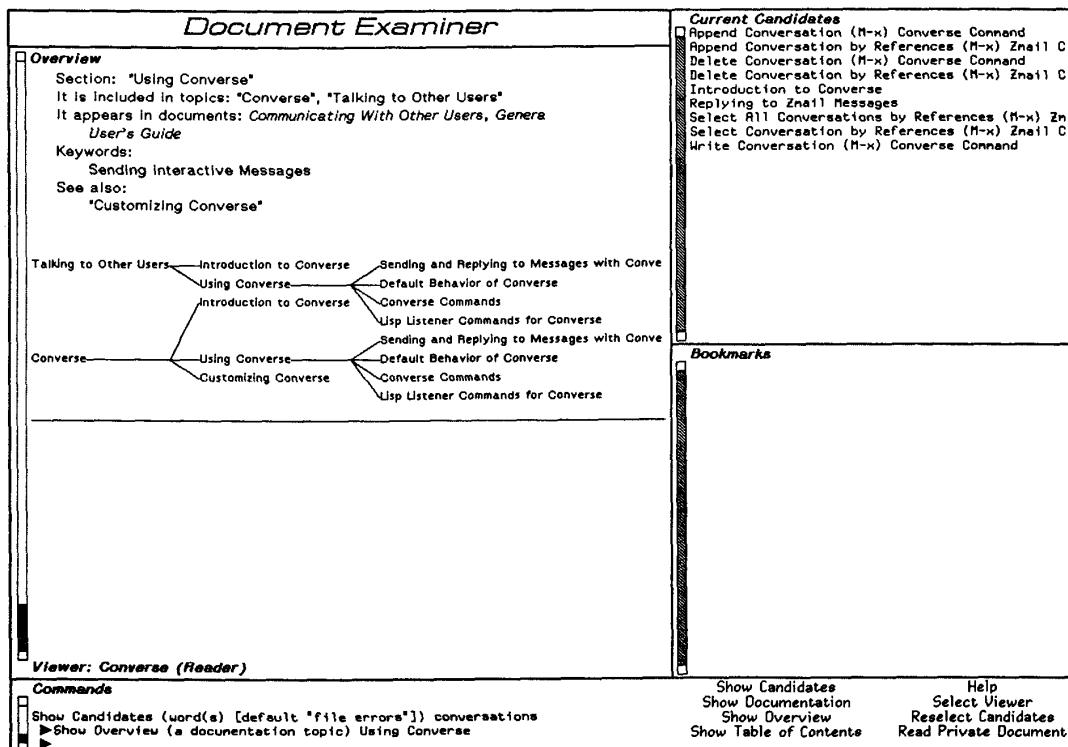
menu and a command interactor area where the user can type commands. (Most commands can be invoked with either the mouse or the keyboard.)

To find relevant sections, users have commands that let them do the on-line equivalent of looking in the index and table of contents for the document set. The set of interesting sections (called *candidates*) is listed in a menu at the top right of the screen. Candidates are *mouse-sensitive*, so a user can click directly on a name to read it or look at how it fits into the document set.

Anyone using a document needs bookmarks to keep track

of important sections. One area of the screen maintains a chronological list of the topics that a user has read in the session. The bookmarks are mouse-sensitive, so the user can reselect a topic read previously.

Topics are rarely independent, and part of using a manual well lies in deciding what to read. Users can request a "peep-hole" display of context for a topic. A temporary display graphs the links between a record and other, immediately related, records. The accompanying figure on Document Examiner's overview window shows an example of an overview for a record that appears in two different documents.



Overview window visible in Document Examiner. The overview describes the topic, "Using Converse." The textual part of the overview shows the keywords associated with this record and the cross-references that it makes to other records. The diagrams show the local context for a record, in what section it appears, and what other sections are nearby in the document.

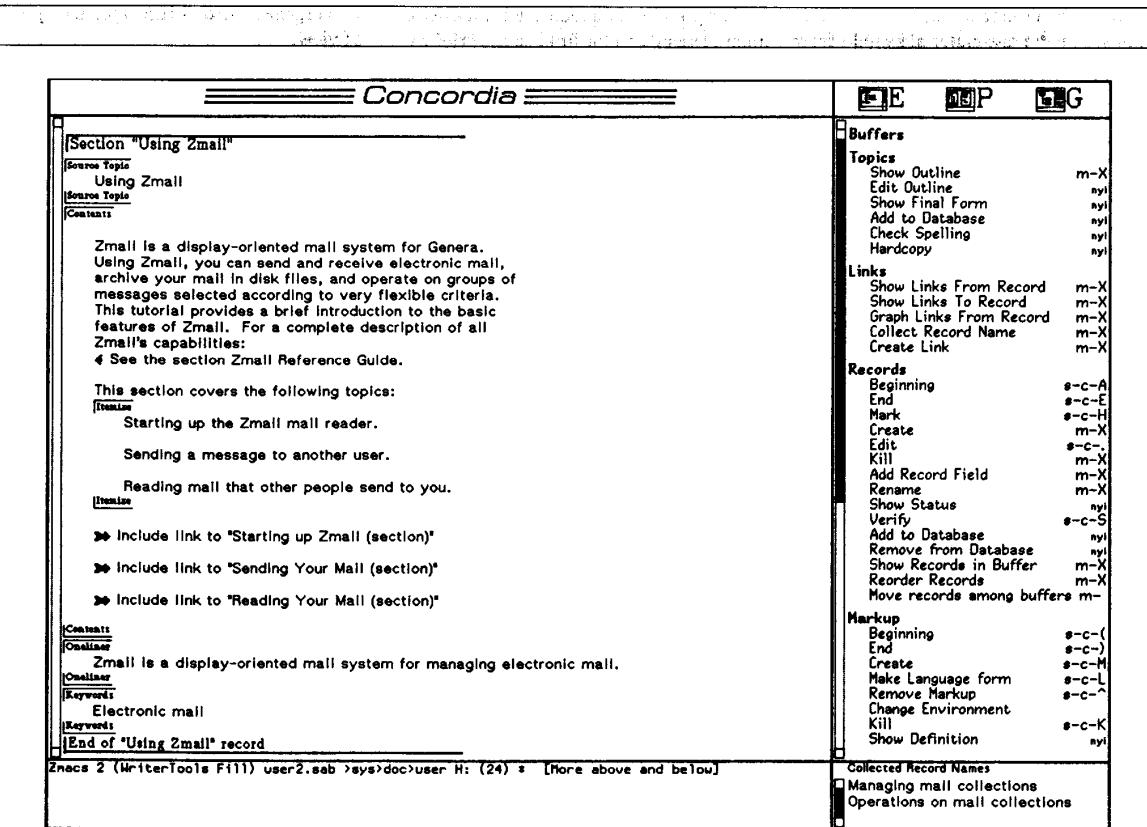


Figure 4. ConEd screen, showing a complete record containing appearance markup and links to other records. Small boxed delimiters (Itemize) show the extent of formatting markup. The vertical spacing and shifted left margin constitute a semblance of the final formatting effects. When this record is processed from a reader's point of view, the records that are the targets of the links are included dynamically as sections within this record.

structural rather than textual elements, writers can't just "type in" new records. Instead, commands are provided for creating records, with a template supplying standard fields and sometimes initial contents for the fields. For example, in creating a new record for a function, Concordia fills the argument list field from information in the compiled object database. A number of validation checks run during record creation ensure against unintentional duplication and spelling errors in definition names.

Creating links. Again, since the links between records are structural rather than textual, ConEd provides commands for creating and changing them. To create a link from one record to another, you place the cursor at the desired origin of the new link, click on the Create Link command,

and then supply the target record. You can type in the target name, but more often would click on it, either on the main screen or in the pane of collected record names in the bottom right corner (Figure 4).

Changing the organization. You change the organization of a document (for example, the order of subsections within a section) by changing the order of the links. You can move a whole subtree of the document simply by moving the link to the root of the subtree. As a result, you can quickly evaluate a number of different organizations for material with little risk of becoming confused or inadvertently deleting material without pasting it back.

Showing relationships between records. ConEd supplies several commands to help

you visualize the structural relationships between records. A table of contents shows the complete subtree below any record to help in visualizing the structure. Other commands show all of the links from the current record or all of the links that point to it. Being able to see the links to a record allows you to manage cross-references to it.

Selecting a record for editing. Although you can scroll around in ConEd buffers or select buffers by name (as in normal text editing), you can also select records directly, by name, for editing. ConEd uses a location database (maintained by the Concordia compiler) to select a record. It ensures that the relevant file is in a buffer (reading it in if necessary), selects the buffer, and positions the cursor so that the requested record is visible on the screen.

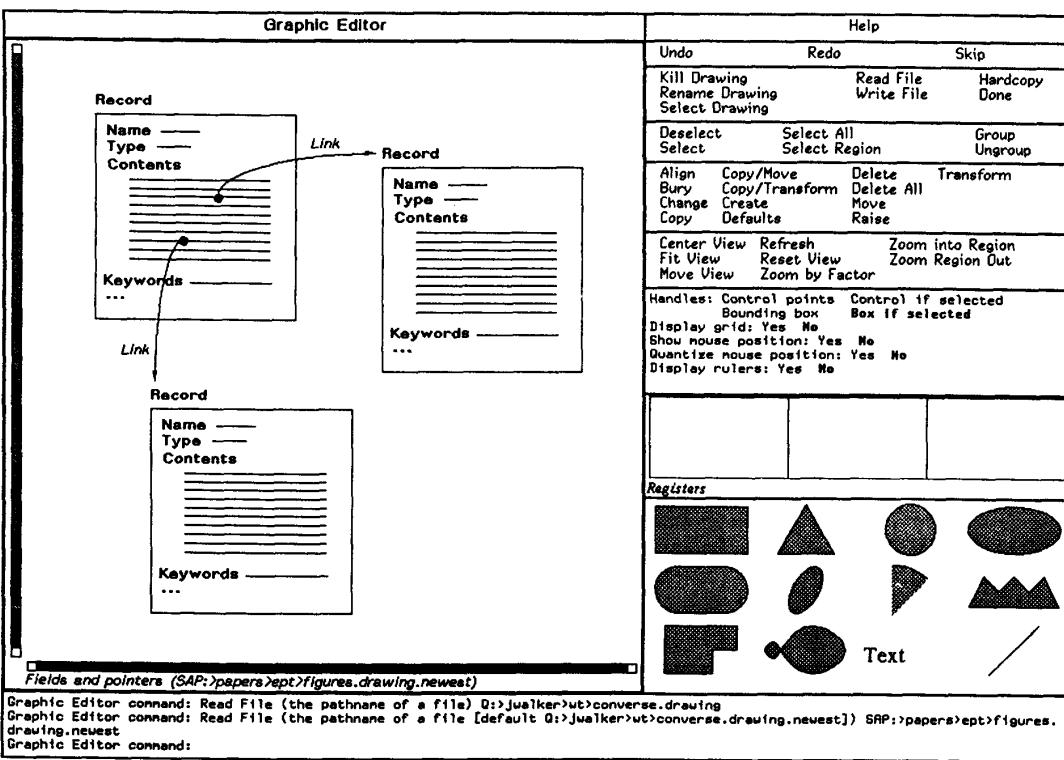


Figure 5. Graphics editor. The main body of the editor contains a figure being edited. The right-hand panel contains several kinds of menus, including a command menu, a shape menu, and a control panel for attributes of the editing. Most of the editing can be performed by typing in commands as well as by mouse selection.

This removes any need to remember file names or perform textual searches to locate material to be edited. Instead, you remember record names—an easier task aided by substantial on-line help. Record name presentations on the screen are mouse-sensitive, so you can click on the name of a record to select it for editing. This makes sequential editing from a marked-up manuscript simple in spite of the underlying modular structure.

Content editing. Documents usually consist of text and pictures. (With computer delivery, other media for documentation such as video, sound, and animation will become feasible as well.) In Concordia, ConEd handles the textual subject matter; an editor for line illustrations handles the graphic subject matter. Using its knowledge about record types, Concordia

switches automatically to the editor appropriate for the subject matter.

The ConEd editor is the top-level framework for handling all aspects of document editing; the graphic editor simply prepares graphical subject matter for incorporation into the records managed by the text editor.

The parts of a record that look like text are text; you use standard text editing commands (as opposed to structure editing commands) to modify them. ConEd has sufficient understanding of the components of text to manipulate words, sentences, and paragraphs (the units of text) as well as characters and lines. (ConEd is an extension of Genera's Zmacs editor, which also has these text handling capabilities.)

The parts of a record that look like pictures are pictures. Concordia's own

graphics editor is an object-oriented editor for line illustrations. It stores drawings as structured descriptions rather than as bit maps to enable flexible editing of structure instead of pixels. To modify one of these pictures, you click on it with the Edit command and Concordia automatically invokes the graphics editor (see Figure 5). Pictures can also be accommodated in a number of other formats, including Lisp graphics programs, bit maps, Postscript, and externally generated pictures (like those from MacDraw).

Appearance editing. In Concordia, a markup language controls the appearance of documents. Unlike most other markup languages, the markup is not embedded text strings.

Markup is the term used for non-procedural descriptions of the generic cat-

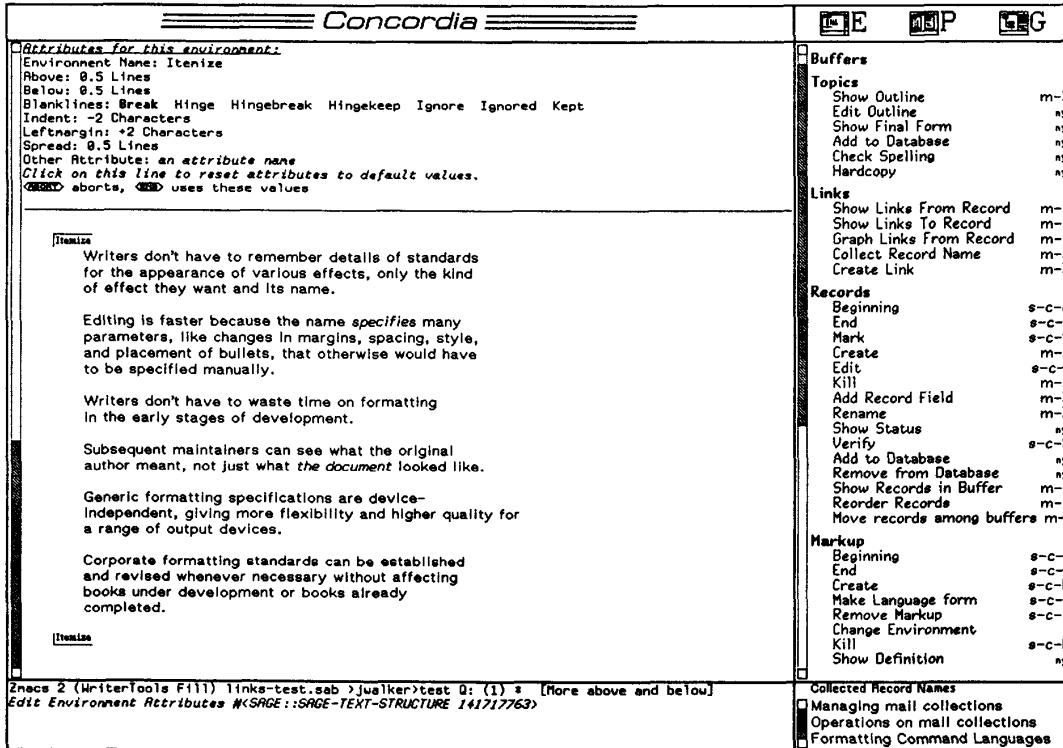


Figure 6. Local markup modification in ConEd. The writer has selected the Itemize markup for modification, bringing up a menu of the attributes involved in its standard definition. After changing and saving the attributes, the changes apply to this one instance, leaving the standard definition unchanged.

egory of information in some region of text. (Figure 4 contains examples of markup.) The markup is delimited visually by small iconic boxes. These delimiters are structural rather than textual, meaning that text editing operations do not apply to them. For example, the text command to delete a line does not delete a delimiter line.

All markup is manipulated by specialized commands, some of which appear in the right-hand menu in the ConEd figures. Commands are used to add markup to existing areas of text and to remove either just the markup or the markup and the relevant area of text. As a result, all of the markup in a record is syntactically correct; no formatting errors can occur later as a result of missing or extra delimiters (common errors with text-based embedded

markup formatters).

ConEd itself shows a semblance of what the final document product will look like. Bold, italic, and fixed-width typefaces appear as such in the editor window, rather than being indicated by font change characters or embedded notation specifying the typeface. The final format, however, is only suggested by indentation, which serves as an aid for checking visually that the markup includes only the intended text.

The markup that controls formatting is backed up by a *book design*, which defines the appearance parameters for all markup used in a book. Markup definitions can be changed globally (in Concordia's book design environment) or locally in ConEd for a particular case. Figure 6 shows the mechanism by which you would use

ConEd to change the appearance specified for a particular instance of a highlighted list. Clicking on one of the markup delimiters brings up a menu of formatting attributes to modify. The modifications are then saved as part of that markup object.

Handling meta-information. WYSIWYG editors require by definition that the document file contain only the information that will appear to the final reader of the document. They require the rest of the information associated with a document to be maintained on paper, informally in unrelated files, or in people's heads. Embedded command formatters usually allow comments as an unstructured way of capturing some of this information. The record structure in Concordia provides

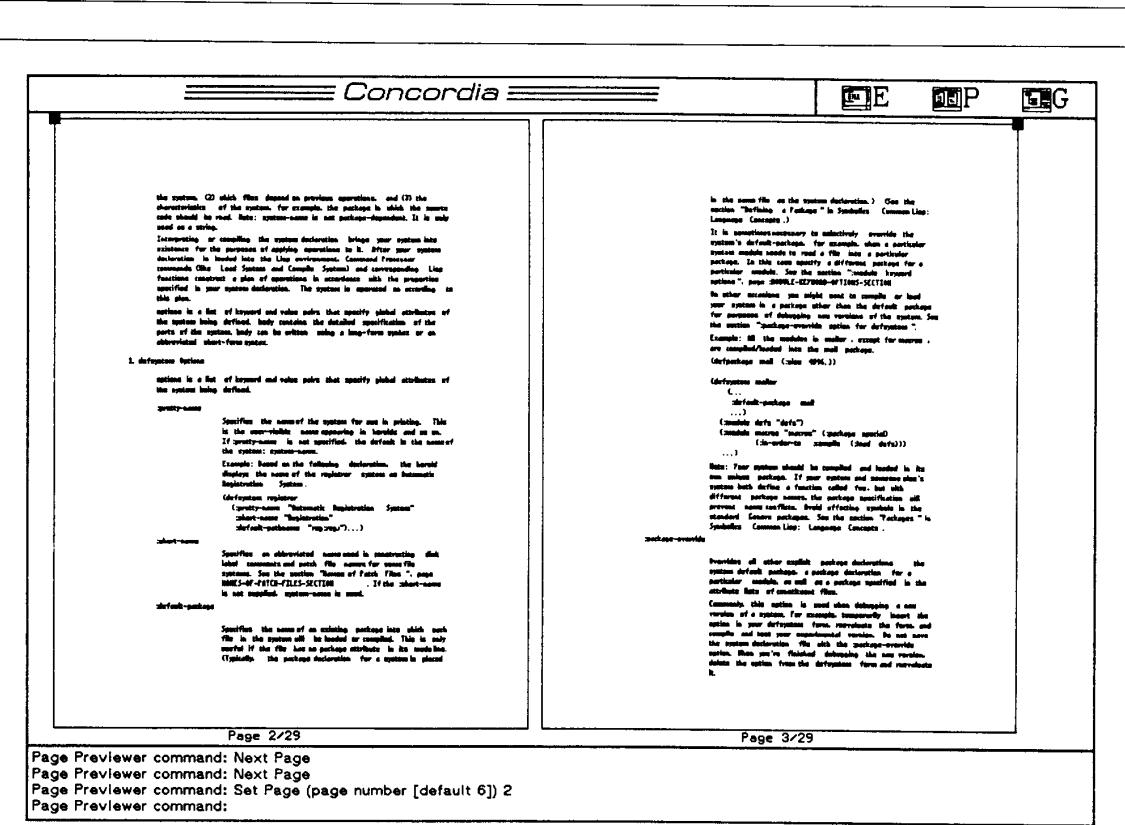


Figure 7. Hardcopy preview. A page previewer shows an exact facsimile of the placement of lines and page breaks. Spacing and fonts used in the preview are different from those on the final output device due to differences in resolution. The placement of words in lines, however, is exactly the same, so this previewer can be used for surveying the progress of the layout. (Note: The text is not intended to be legible because this is used for design purposes, not proofreading.)

fields for storing accessory information (for example, keywords, auditing information, and notes) and, in some cases, for processing it.

One kind of meta-information stored in a record is its verification status. Concordia keeps track of whether records have been formatted since being changed or changed without being installed in the database. This status information is used by various commands to help writers keep track of their workload.

Viewing and reviewing documents

At different points in the document life cycle, writers need different ways of look-

ing at their work. Concordia provides a number of ways to view a document.

Seeing the reader's viewpoint. The semiblance editing in ConEd gives a good indication of the formatting structure within a record. It is often necessary, however, to look at a record from the perspective of a reader, with its links expanded. ConEd has a facility for *formatting on demand* that shows a record formatted on the screen as the eventual reader of it would see it in Document Examiner or on paper.

Local hardcopy. You can produce hardcopy of any topic (a record and its expansions) in the database. Whether or not to use paper is a question of personal preference, since paper is not required for any stage of development.

Preview. During final production of a paper manual, you must consider the placement of ink on paper. For this stage, Concordia provides a page formatter that shows on the screen a miniature but exact facsimile of how the document will appear when printed (see Figure 7). You can identify badly placed page breaks and poor formatting decisions (and then fix the source files) without having to print out any paper copy at all. The text in this previewer is not supposed to be legible; it is used for proofing the overall layout, not the text.

Final hardcopy. As the last stage in production, Concordia produces print masters for each book, expressed in Postscript.⁹ The masters contain everything needed to print a book (front matter, table

of contents, index, figures, running heads), leaving very little manual work in final production.

Production

Managing the files for a large document set consisting of one or more books

requires effective configuration management tools, since manually managing the files involved in various versions of a large document set is very difficult. Concordia uses the system configuration tools (SCT) in Genera to address these information management requirements.

Systems. SCT provides the mechanism

for specifying a document set as a *system*. A system is a formal data structure that manages a set of files and defines the operations available for those files, such as editing, formatting, updating, and distributing.

Incremental update. Large documents are created by teams of writers and

Formatter command languages

Before WYSIWYG editing and desktop publishing, people lucky enough to have access to time-sharing computer systems used embedded-command batch formatters for their documents. The documents were plain text files with formatter commands intermingled with the text; a flag character indicated the presence of a command.

The following shows examples of three major families of such formatters, the dot-in-column-one formatters (like Runoff), the \ formatters (like TEX¹), and the @ formatters (like Scribe²).

```
.br  
.s 1  
.lm 5  
.i -2  
* If call-next-method is used in an :around method ...
```

```
\begin{list}  
\item{\{bull\}}  
If {\bf \{bf} call-next-method} is used in an {\bf \{bf :around}}  
method...
```

```
@begin[itemize]  
If @b[call-next-method] is used in an @b[:around] method...
```

Many of these formatters had advanced macro and programming capabilities and are still in use today because they have a number of advantages over the WYSIWYG approaches.

There are two basic classes of command languages for specifying document formatting—procedural languages and declarative languages. The procedural languages (like Runoff) require writers to specify all formatting directly, in effect “programming” their documents. Some allow macros as a way of masking the programming and simplifying the document sources. The declarative languages, of which Scribe is the best example, specify what the text is in an abstract sense and leave the exact specifications about appearance to a book design database.

Declarative languages introduce a level of abstraction that separates the kind of formatting wanted from the exact details of how to produce it. Thus, such languages are *generic*, specifying documents in such a way that they can be produced for a variety of different output devices, from typewriters to typesetters. This approach is the very antithesis of simple WYSIWYG, which uses the screen as an exact representation of a single final paper result.

In a generic markup language, writers specify their *intent* for a particular part of the document without specifying any of the details of its appearance. For example, the generic name for a list with bulleted paragraphs might be “highlighted-list.”

This name specifies only the nature of the effect wanted, not the details on how to achieve it.

Generic specification has a number of advantages over simple WYSIWYG editing or highly procedural formatting languages:

- Writers don't have to remember details of the local conventions for the appearance of various effects, only the kind of effect they want and its name.
- Editing is faster because the name implies many parameters, like changes in margins, spacing, style, and placement of bullets, that otherwise would have to be specified manually.
- Writers don't have to waste time on formatting in the early stages of development.
- Subsequent maintainers can see what the original author meant, not just what the document looked like.
- Generic formatting specifications are device-independent, giving more flexibility and higher quality for a range of output devices.
- Corporate formatting standards can be established and revised whenever necessary without affecting books under development or books already completed.

To assist in hand-formatting a document, WYSIWYG editors provide style sheets and procedural formatting languages provide macros. These approaches do not offer the same power as a generic description language.

With the advantages of generic markup languages, you might wonder why simple WYSIWYG editors have such power in the marketplace. Undoubtedly there are many answers to this question.³ Aside from WYSIWYG's roots in the typing world and a general lack of understanding of document abstractions, one of the answers lies in the convenience and aesthetics of WYSIWYG editing itself.

Most embedded markup languages make the document source file appear complicated by requiring writers to insert formatting codes into their text. Errors are common, due to the difficulty of embedding codes correctly, and sorting out errors can be maddening, as there is rarely any debugging support. Such formatting systems make the document source difficult to read and hence make the job of working on it unpleasant, in spite of the advantages.

References

1. D.E. Knuth, *The TeXbook*, Addison-Wesley, Reading, Mass., 1984.
2. B.K. Reid, *Scribe: A Document Specification Language and Its Compiler*, Carnegie Mellon University, Pittsburgh, Penn., Dec. 1980.
3. J.H. Coombs, A.H. Renear, and S.J. DeRose, “Markup Systems and the Future of Scholarly Text Processing,” *Comm. ACM*, Nov. 1987, pp. 933-947.

engineers whose work can be highly interdependent; sections from one book need to refer to those in another. Using Concordia, you can add your changes to the database daily (or more often), which makes those changes available immediately to anyone else working on the same project.

Version and configuration control. SCT records the source and update files that constitute any particular version of a document. As a result, document versions and software versions are coordinated automatically. A particular system version can be distributed and its files marked to protect them against deletion.

Evaluation

The document development methodology in Concordia has been used in-house at Symbolics since late 1983. The documentation group has consisted of eight writers (on average), one editor, one supervisor, and one person responsible for production, each equipped with a Symbolics workstation and software. In this four-year period, the group has published three major editions of the Symbolics document set, ranging in size from 2500 pages (1984) to over 7500 pages (1987). Each new edition was completely reprinted. Several minor releases intervened between the major releases, each with release notes and sometimes newly added documents.

The writing group members are not the only users of Concordia. Many software developers also use Concordia for organizing design documents and for first-draft reference documentation.

Our approach to document development has been particularly successful in the following areas:

- **Fast prototyping.** New documents based on existing material can be put together in days. New organizations for existing documents can be tried out quickly and maintained in parallel with the original.

- **On-line delivery.** A single document database is used for both on-line delivery and paper manuals; both media deliver exactly the same documents. With our display hardware and Document Examiner interface, we have found on-line delivery an acceptable alternative to paper.

- **Quality enhancement.** Since the in-house engineering community has access to the document database, documents are actually in use during development. As a

result, users can report errors and usability problems as documentation bugs via electronic mail. Minor revisions are immediately available.

- **Maintenance.** Writers can update documents easily by replacing erroneous records or by adding new ones, and updated records are distributed electronically to customers as part of minor releases. The writing staff can respond quickly and easily to problem reports because changes do not result in change pages. They manage updates with the same configuration tools as used for software changes.

We plan to continue using Concordia for developing documentation at Symbolics, extending it as our needs expand. We see a number of areas needing further research and exploration:

- (1) **Project support.** Documentation is produced by groups of people coordinating their efforts with other groups of people. We need to address further technical aspects of this coordination.

- (2) **Understanding modular writing.** We need more research to understand the difficulties inherent in technical communication, particularly in the rhetoric of modular writing.

This approach is feasible for producing large-scale documentation. Using it, our writers have become highly productive in a demanding development environment. □

References

1. D.E. Engelbart, "Authorship Provisions in AUGMENT," *Intellectual Leverage: The Driving Technologies*, IEEE Spring Compcon84, 1984, pp. 465-472.
2. J.H. Walker, "Symbolics Sage: A Documentation Support System," *Intellectual Leverage: The Driving Technologies*, IEEE Spring Compcon84, 1984, pp. 478-483.
3. J.H. Walker et al., "Symbolics Genera Programming Environment," *IEEE Software*, Nov. 1987, pp. 36-45.
4. G. James, *Document Databases*, van Nostrand Reinhold, New York, 1985.
5. J. Conklin, "Hypertext: An Introduction and Survey," *Computer*, Sept. 1987, pp. 17-41.
6. R. Furuta, J. Scofield, and A. Shaw, "Document Formatting Systems: Survey, Concepts, and Issues," *Computing Surveys*, Vol.14, 1982, pp. 417-472.
7. J.H. Walker, "Document Examiner: Delivery Interface for Hypertext Documents," *Proc. Hypertext '87 Workshop*, Chapel Hill, N.C., Nov. 1987, pp. 307-323.
8. J.H. Walker and R.L. Bryan, "An Editor for Structured Technical Documents," in *Protext IV: Proc. 4th Int'l Conf. on Text Processing Systems*, Boole Press, Dublin, Ireland, 1987.
9. Adobe Systems Inc., *PostScript Language Manual*, 1st ed., Palo Alto, Calif., 1984.



Janet H. Walker is a principal member of the technical staff at Symbolics, Inc. Her research interests include user interfaces for software and document development environments.

Walker received the BSc degree from Carleton University, Canada, and the AM and PhD in cognitive psychology from the University of Illinois at Urbana-Champaign in 1974. She is a member of ACM and the Computer Society of the IEEE.

Readers may write to the author at Symbolics, Inc., 11 Cambridge Center, Cambridge, MA 02142; jwalker@symbolics.com is her electronic mail address.